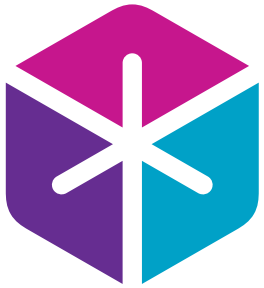
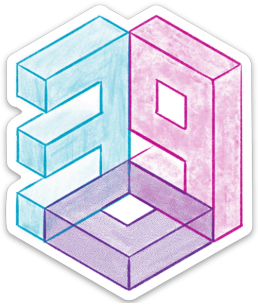




https://introml.mit.edu/

6.390 Intro to Machine Learning

Lecture 6: Neural Networks II

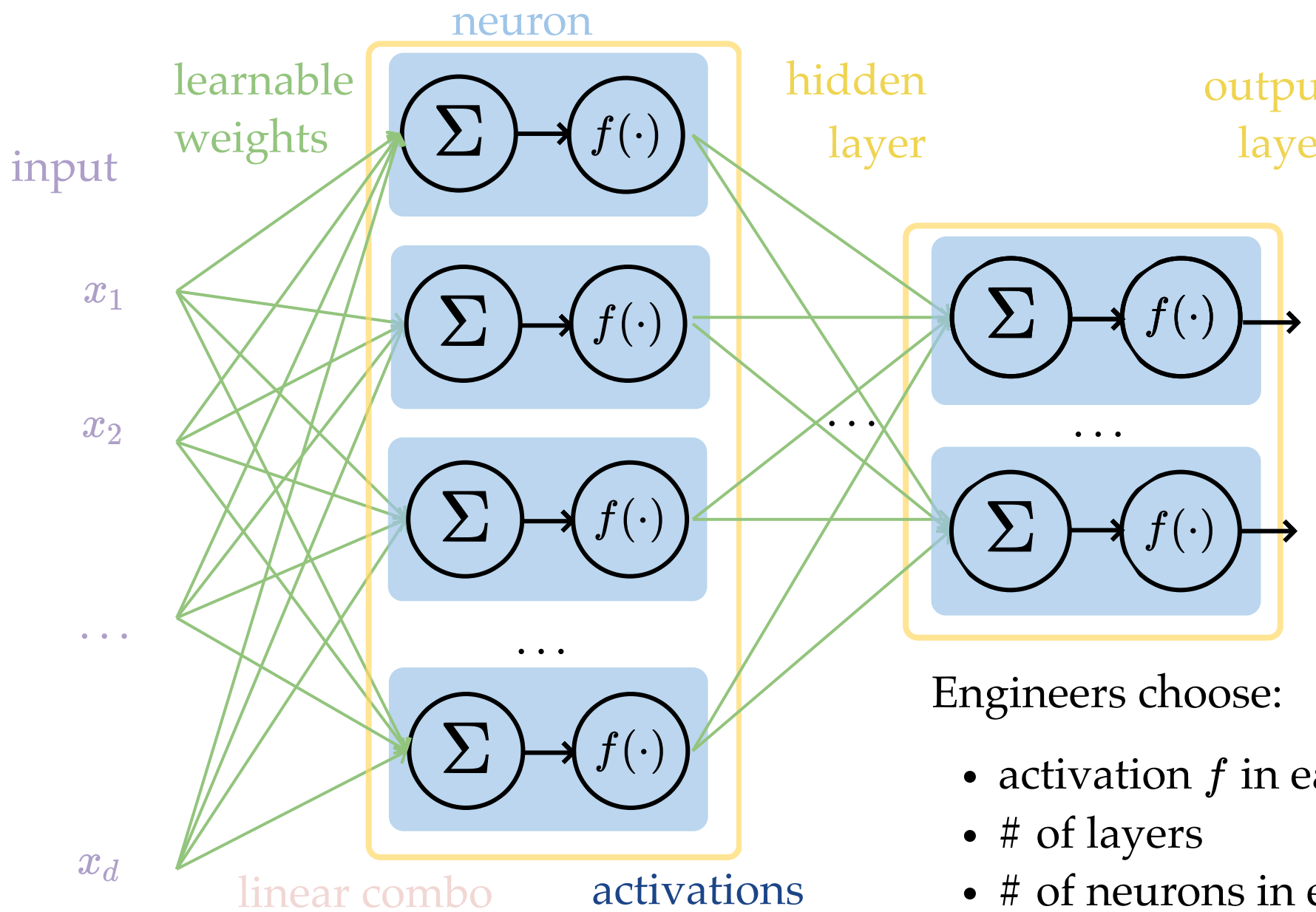
Shen Shen

Mar 9, 2026

3pm, Room 10-250

Slides and Lecture Recording

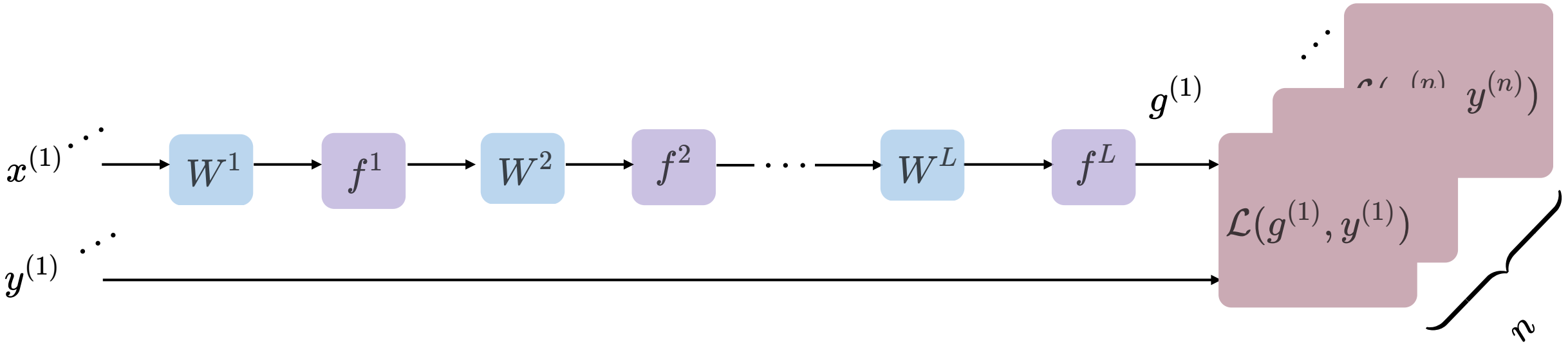
Recap a (fully-connected, feed-forward) neural network (aka, multi-layer perceptrons)



Engineers choose:

- activation f in each layer
- # of layers
- # of neurons in each layer

`Recap` Forward pass: evaluate, *given* the current parameters



- the model outputs $g^{(i)} = f^L\left(\ldots f^2\left(\ f^1(\mathbf{x}^{(i)}; \mathbf{W}^1); \mathbf{W}^2\right); \ldots \mathbf{W}^L\right)$
- the loss incurred on the current data $\mathcal{L}(g^{(i)}, y^{(i)})$
- the training error $J = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(g^{(i)}, y^{(i)})$

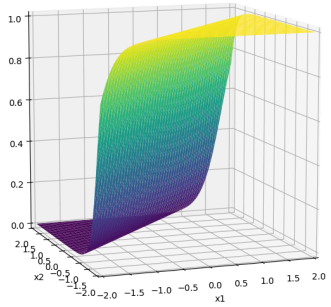
linear combination

(nonlinear) activation

loss function

Recap

$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$



$f(\cdot)$ identity function

x_1

x_2

1

5 -5 1

1 5 -5

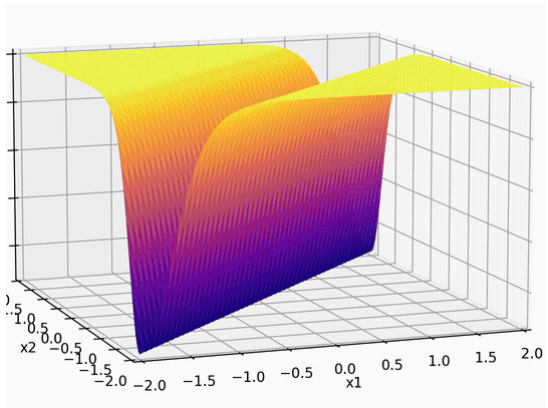
W^1

$f = \sigma(\cdot)$

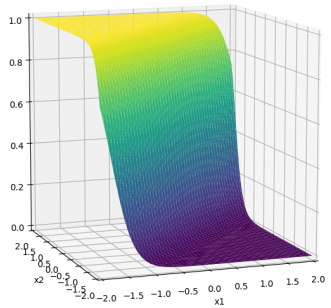
-3

-3

W^2



$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$



$-3(\sigma_1 + \sigma_2)$

(The demo won't embed in PDF. The direct link below gets to the demo.)

https://shenshen.mit.edu/demos/2layers.html

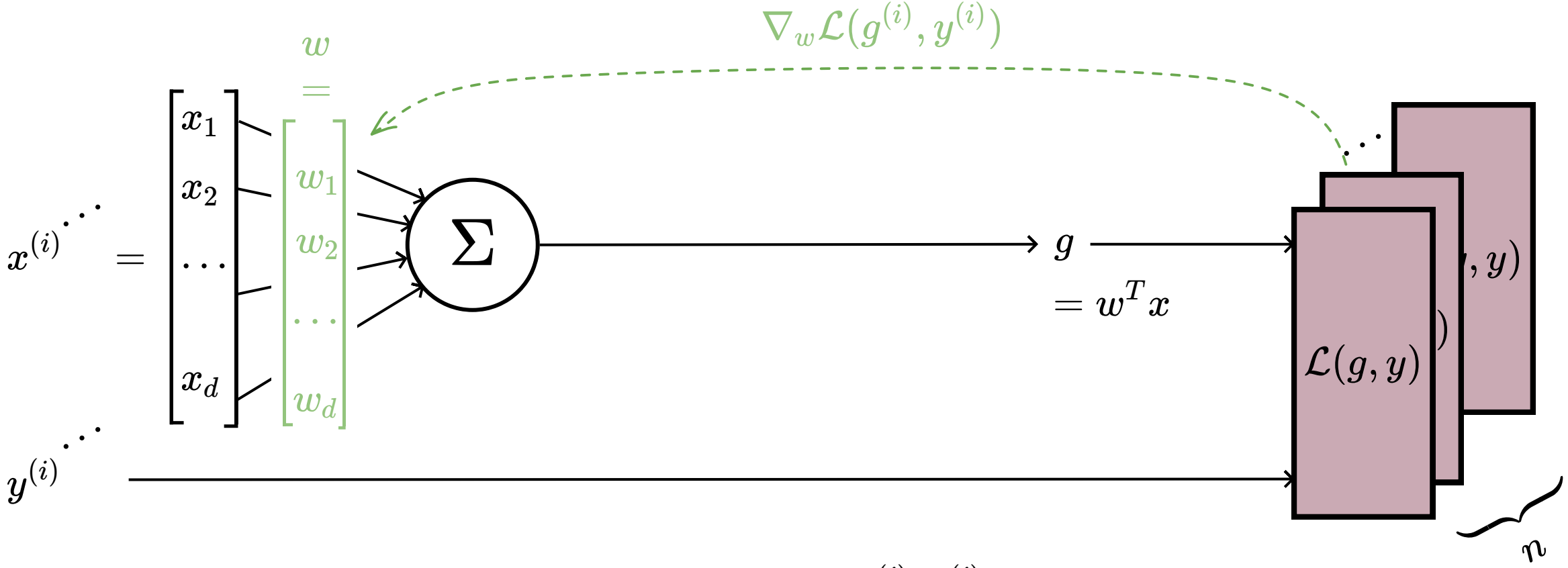
(The demo won't embed in PDF. The direct link below gets to the demo.)

https://shenshen.mit.edu/demos/nn/playground/?embed

Outline

- Backward pass (to learn parameters/weights)
  - Backpropagation (gradient descent & the chain rule)
  - Recursive reuse of computation
  - Practical gradient issues and remedies

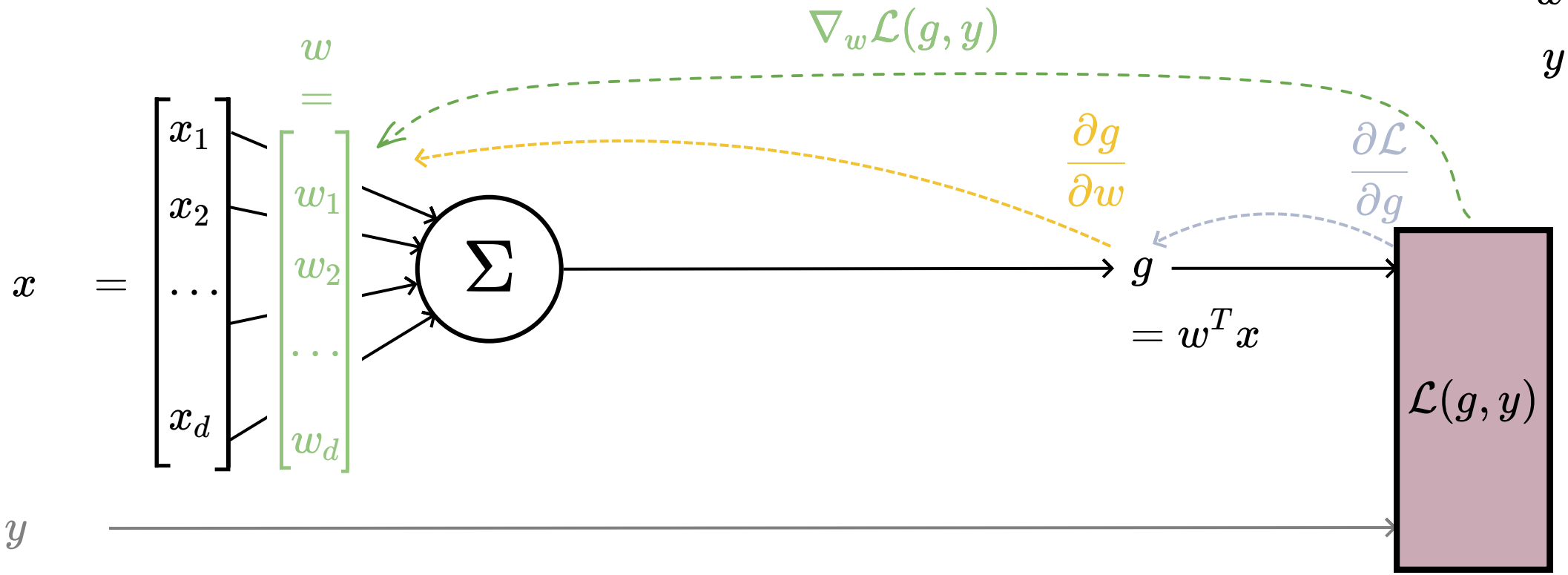
stochastic gradient descent to learn a linear regressor



- Randomly pick a data point $(x^{(i)}, y^{(i)})$
- Evaluate the gradient $\nabla_w \mathcal{L}(g^{(i)}, y^{(i)})$
- Update the weights $w \leftarrow w - \eta \nabla_w \mathcal{L}(g^{(i)}, y^{(i)})$

for simplicity, say training data set is just (x, y) and squared loss

$x \in \mathbb{R}^d$
$w \in \mathbb{R}^d$
$y \in \mathbb{R}$



$$\nabla_w \mathcal{L}(g, y) = \frac{\partial \mathcal{L}(g, y)}{\partial w} = \frac{\partial g}{\partial w} \cdot \frac{\partial \mathcal{L}}{\partial g} = \frac{\partial g}{\partial w} \cdot \frac{\partial (g - y)^2}{\partial g} = \frac{\partial (w^T x)}{\partial w} \cdot 2(g - y) = x \cdot 2(g - y)$$

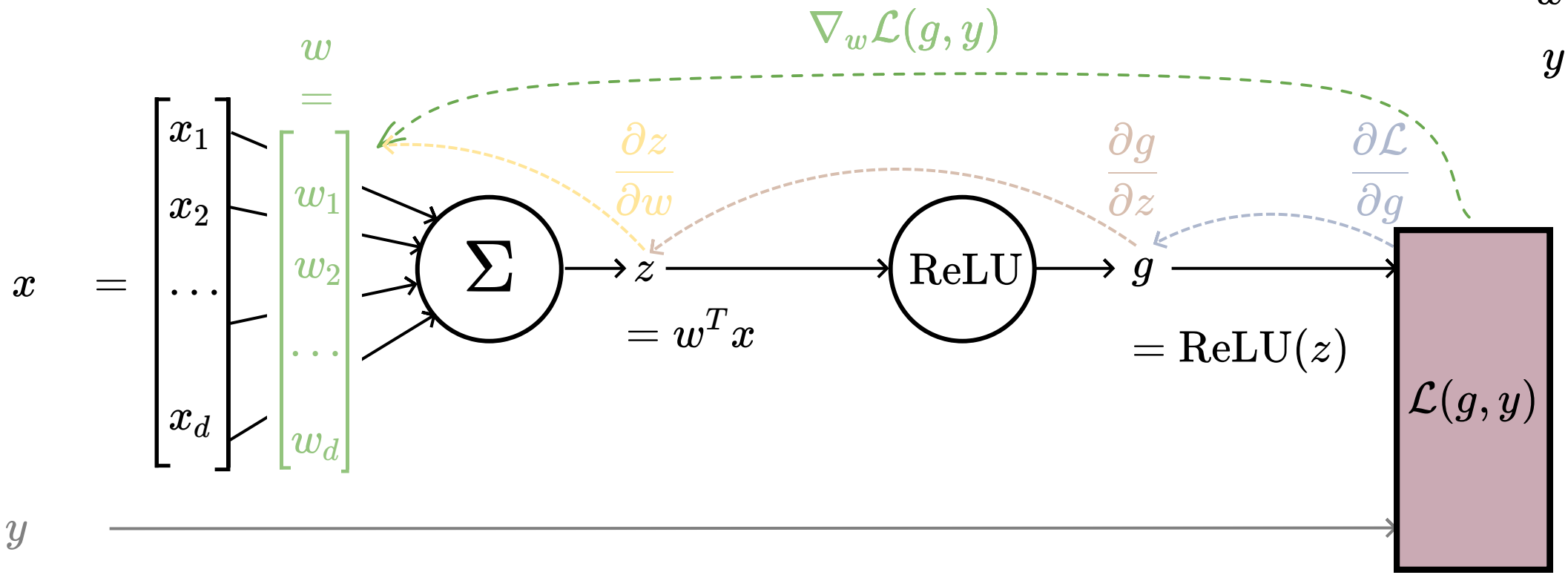
example on the blackboard

Slightly more interesting:

$x \in \mathbb{R}^d$

$w \in \mathbb{R}^d$

$y \in \mathbb{R}$



$$\nabla_w \mathcal{L}(g, y) = \frac{\partial \mathcal{L}(g, y)}{\partial w} = \frac{\partial z}{\partial w} \cdot \frac{\partial g}{\partial z} \cdot \frac{\partial \mathcal{L}}{\partial g} = \frac{\partial z}{\partial w} \cdot \frac{\partial g}{\partial z} \cdot \frac{\partial (g - y)^2}{\partial g} = x \cdot \frac{\partial [(\text{ReLU}(z))]}{\partial z} \cdot 2(g - y)$$

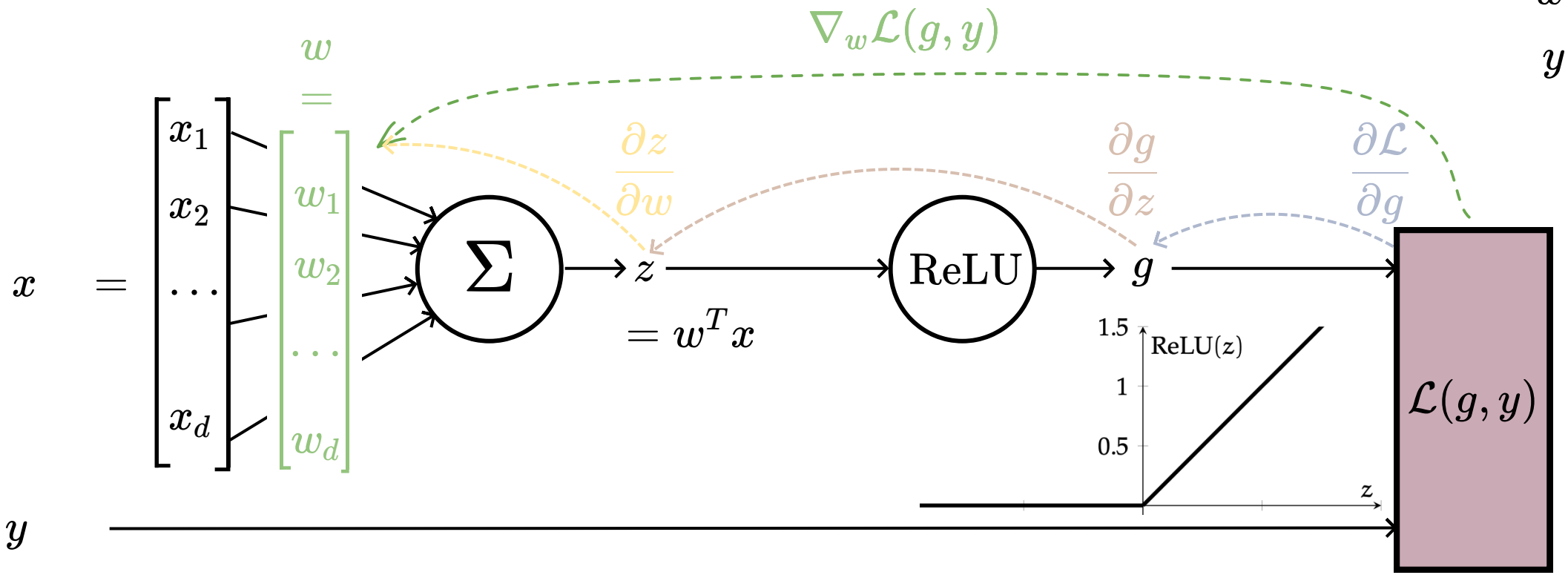
example on the blackboard

Slightly more interesting:

$x \in \mathbb{R}^d$

$w \in \mathbb{R}^d$

$y \in \mathbb{R}$



$$\nabla_w \mathcal{L}(g, y) = x \cdot \frac{\partial[(\text{ReLU}(z))]}{\partial z} \cdot 2(g - y) = \begin{cases} 0, & \text{if } z < 0 \\ 2\,x\,(g - y), & \text{otherwise} \end{cases}$$

$$= \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{otherwise} \end{cases}$$

example on the blackboard

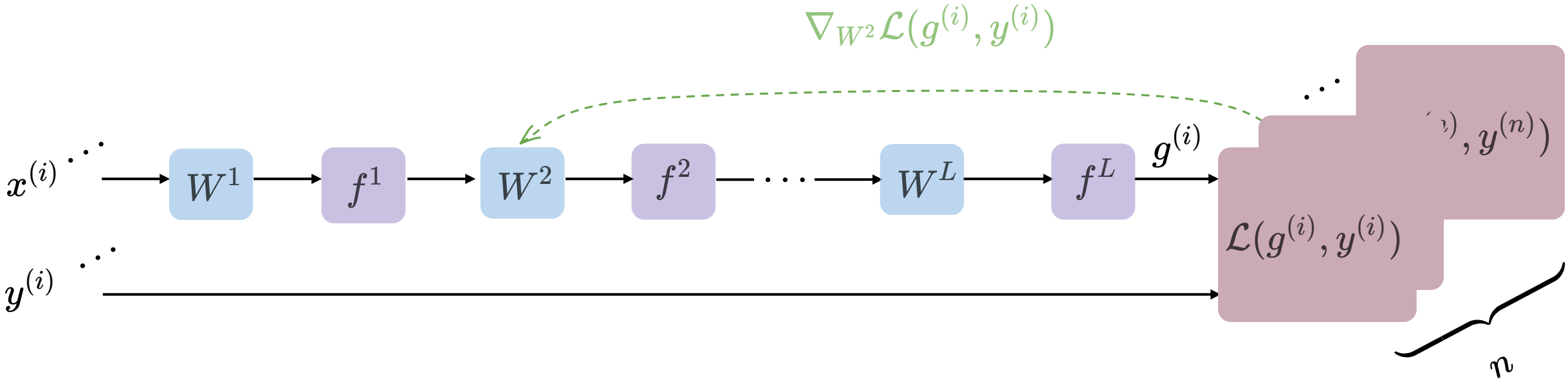
Outline

- Backward pass (to learn parameters/weights)
  - Backpropagation (gradient descent & the chain rule)
  - Recursive reuse of computation
  - Practical gradient issues and remedies

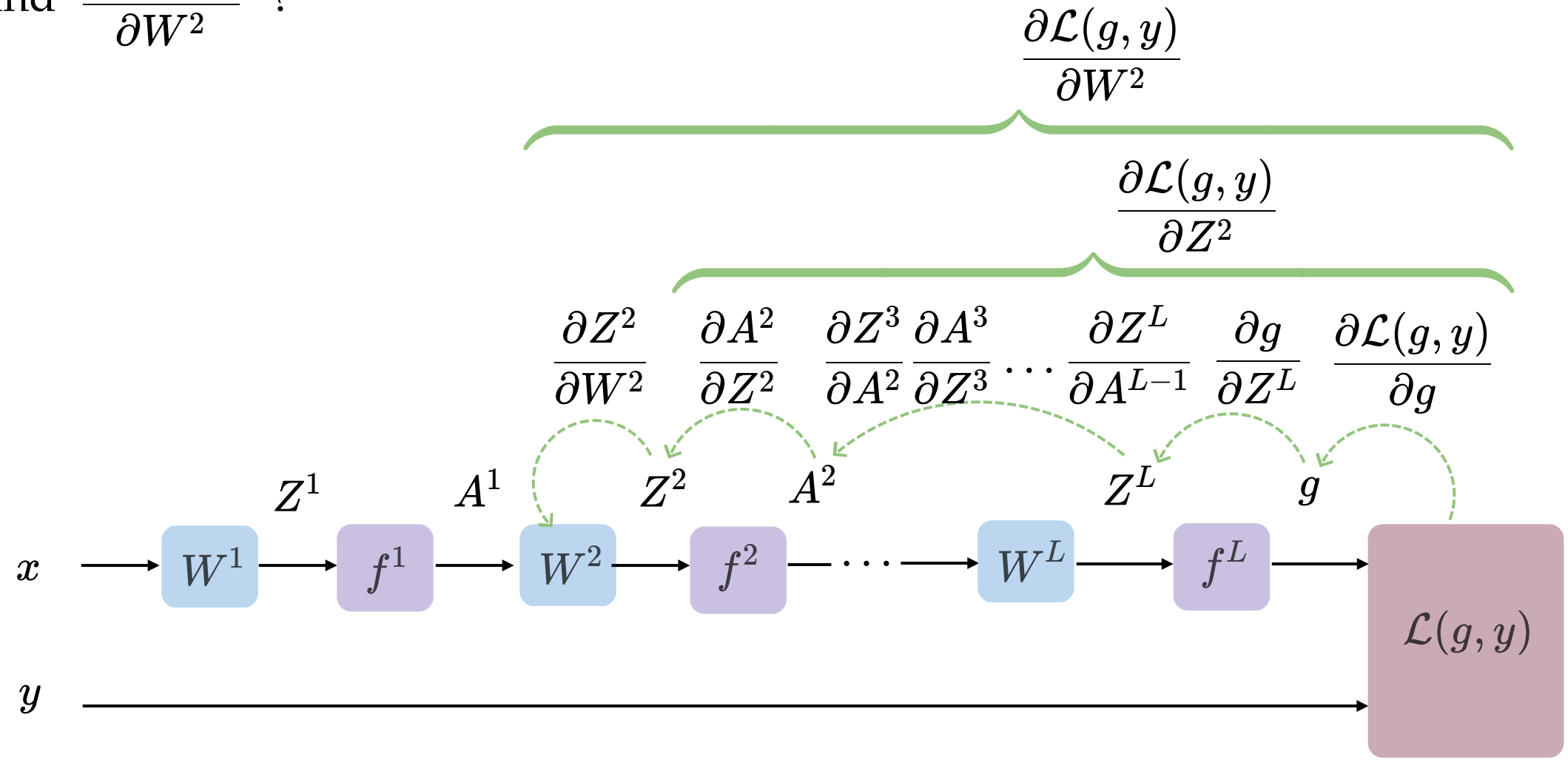
Backward pass: run SGD to update all parameters

e.g. to update W^2

- Randomly pick a data point $(x^{(i)}, y^{(i)})$
- Evaluate the gradient $\nabla_{W^2}\mathcal{L}(g^{(i)}, y^{(i)})$
- Update the weights $W^2 \leftarrow W^2 - \eta\nabla_{W^2}\mathcal{L}(g^{(i)}, y^{(i)})$

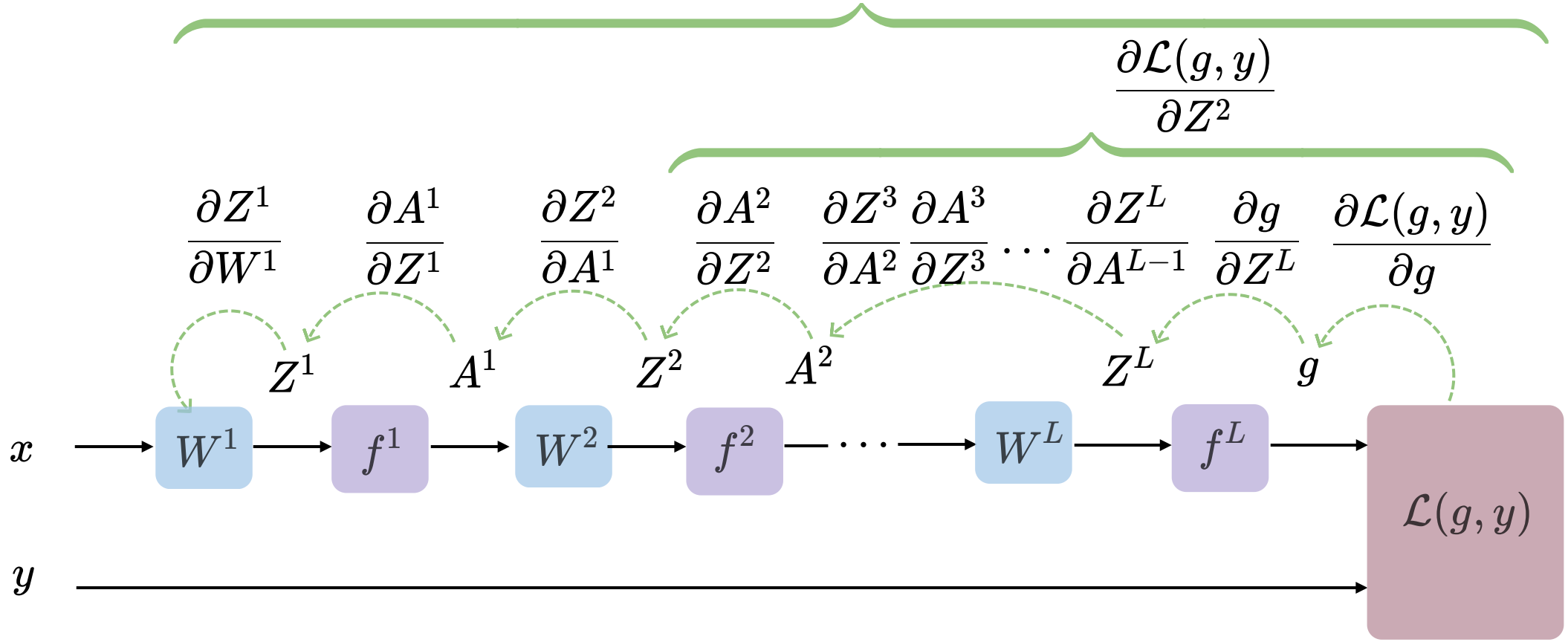


how to find $\frac{\partial \mathcal{L}(g,y)}{\partial W^2}$?



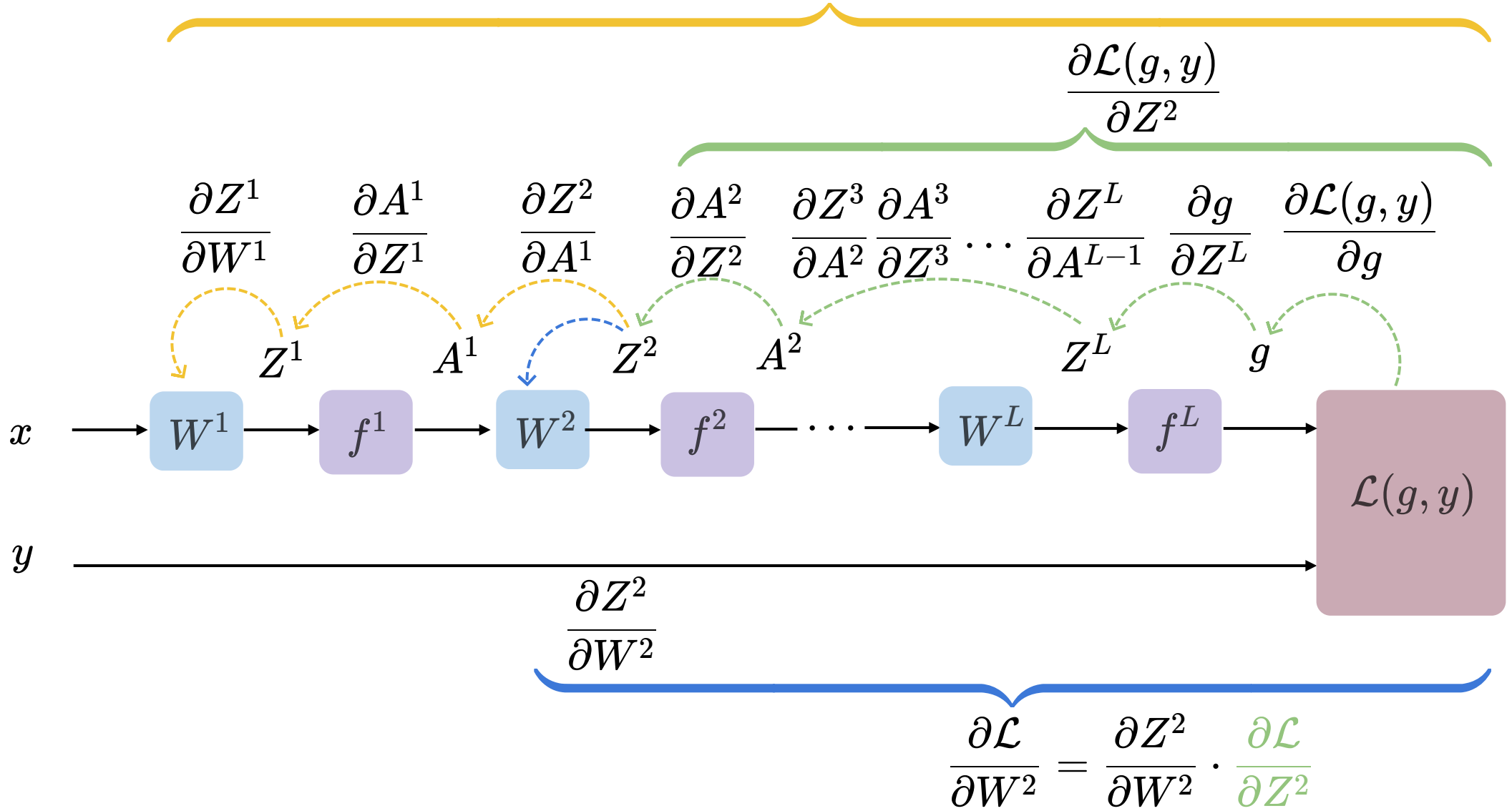
how to find $\frac{\partial \mathcal{L}(g,y)}{\partial W^1}$

Now $\frac{\partial \mathcal{L}(g,y)}{\partial W^1}$



backpropagation: reuse of computation

$$\frac{\partial \mathcal{L}}{\partial W^1} = \frac{\partial Z^1}{\partial W^1} \cdot \frac{\partial A^1}{\partial Z^1} \cdot \frac{\partial Z^2}{\partial A^1} \cdot \frac{\partial \mathcal{L}}{\partial Z^2}$$



$$\frac{\partial \mathcal{L}}{\partial W^2} = \frac{\partial Z^2}{\partial W^2} \cdot \frac{\partial \mathcal{L}}{\partial Z^2}$$

Training a neural network: the full loop

Initialize all weights $W^1, W^2, \ldots, W^L$ randomly

Repeat until stopping criterion:

1. **Forward pass**: for each data point, compute $Z^1, A^1, Z^2, A^2, \ldots, g^{(i)}$
2. **Evaluate loss**: for each data point, compute $\mathcal{L}(g, y)$
3. **Backward pass**: pick a data point, compute $\nabla_{W^\ell} \mathcal{L}(g^{(i)}, y^{(i)})$ for all $\ell = L, L-1, \ldots, 1$ via the chain rule

   (reuse intermediate results → backpropagation)

   e.g., $$\frac{\partial \mathcal{L}}{\partial W^1} = \frac{\partial Z^1}{\partial W^1} \cdot \frac{\partial A^1}{\partial Z^1} \cdot \frac{\partial Z^2}{\partial A^1} \cdot \frac{\partial \mathcal{L}}{\partial Z^2}$$

   $$\frac{\partial \mathcal{L}}{\partial W^2} = \frac{\partial Z^2}{\partial W^2} \cdot \frac{\partial \mathcal{L}}{\partial Z^2}$$

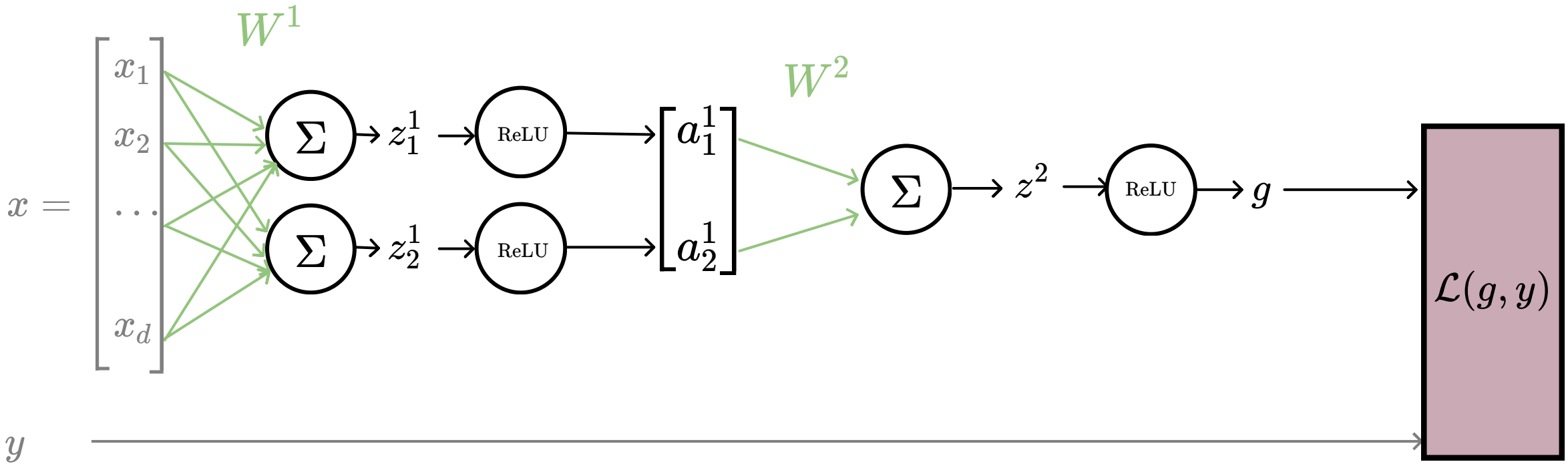
4. **Update**: $W^\ell \leftarrow W^\ell - \eta \nabla_{W^\ell} \mathcal{L}$ for all ℓ

In practice, we use mini-batches rather than a single point.

Outline

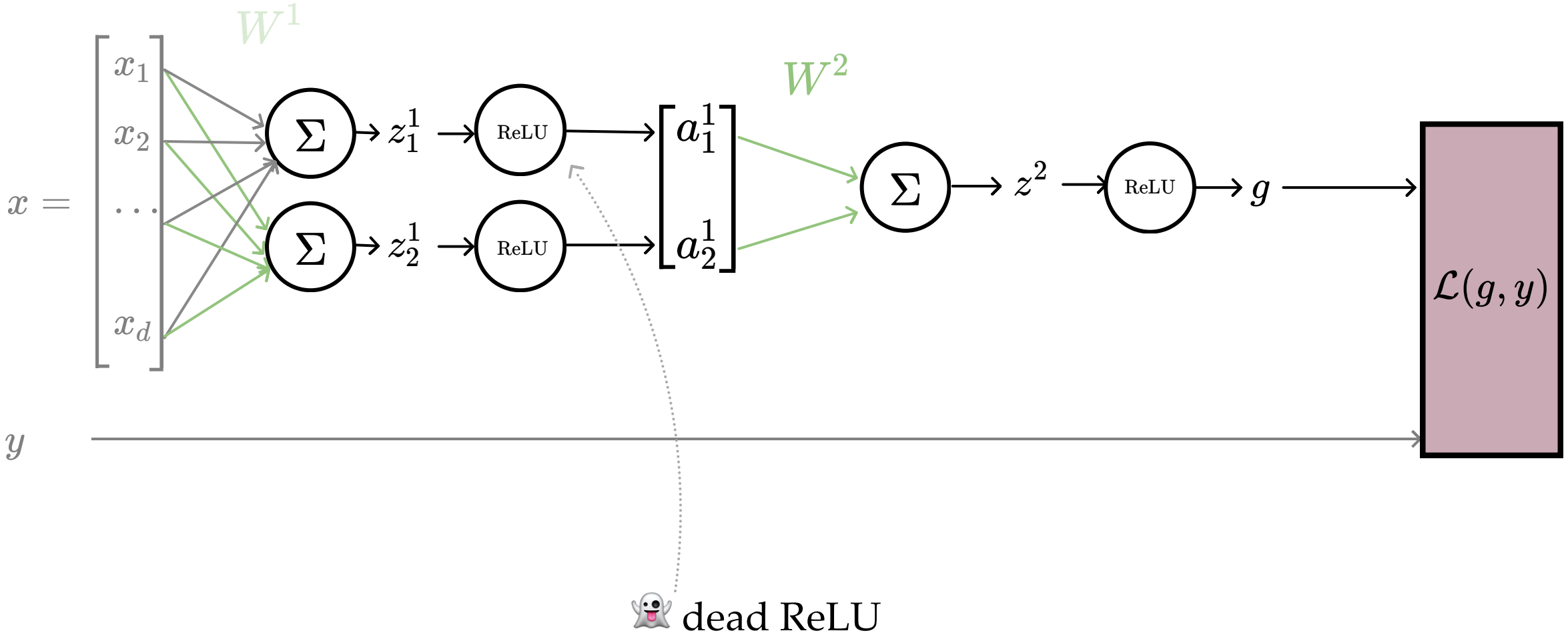
- Backward pass (to learn parameters/weights)
  - Backpropagation (gradient descent & the chain rule)
  - Recursive reuse of computation
  - Practical gradient issues and remedies

More neurons, more layers



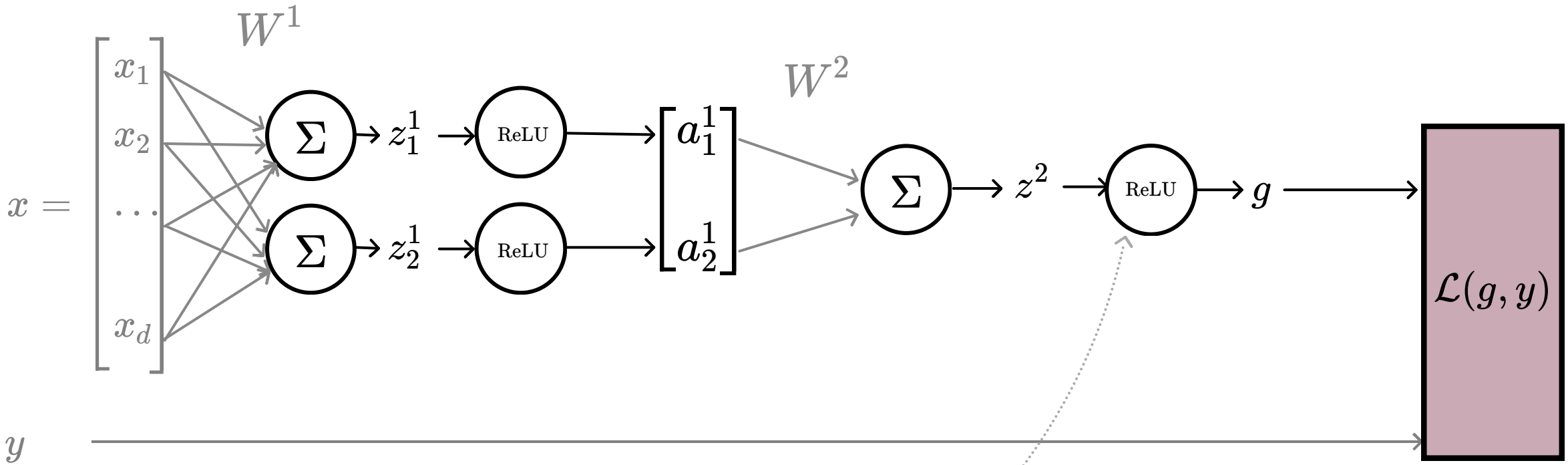
example on the blackboard

if $z^2 > 0$ and $z_1^1 < 0$, some weights (grayed-out ones) won't get updated



example on the blackboard

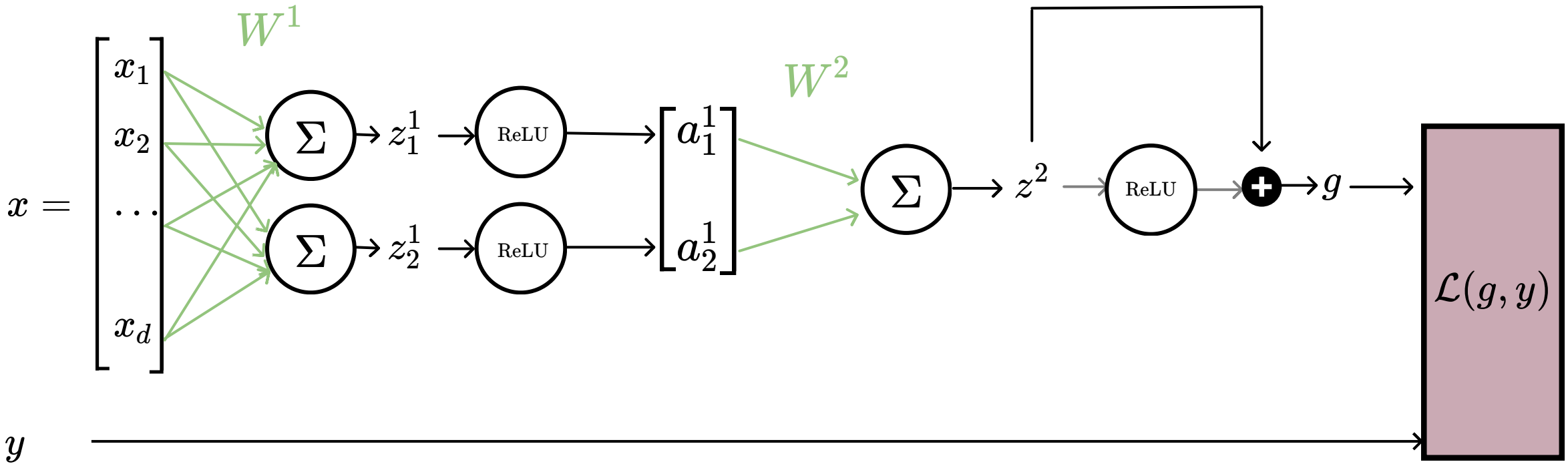
if $z^2 < 0$, no weights get updated



👻 dead ReLU

example on the blackboard

Residual (skip) connection:



Now, $g = z^2 + \mathrm{ReLU}(z^2)$

even if $z^2 < 0$, with skip connection, weights in earlier layers can still get updated

example on the blackboard

Vanishing and Exploding Gradients

Backpropagation computes gradients via the chain rule — a product of many factors:

e.g. $$\frac{\partial \mathcal{L}}{\partial W^1} = \frac{\partial Z^1}{\partial W^1} \cdot \underbrace{\frac{\partial A^1}{\partial Z^1} \cdot \frac{\partial Z^2}{\partial A^1} \cdots \frac{\partial A^{L-1}}{\partial Z^{L-1}} \cdot \frac{\partial Z^L}{\partial A^{L-1}}}_{L-1 \text{ layers of multiplicative factors}} \cdot \frac{\partial g}{\partial Z^L} \cdot \frac{\partial \mathcal{L}}{\partial g}$$

If each factor is small, their product shrinks very quickly with depth, little learning.

If factors are large (magnitude> 1), gradients explode, also problematic.

Many practical remedies: residual connection, gradient clipping

Gradient clipping: If $\|\nabla\| > \tau$, rescale: $\nabla \leftarrow \tau \frac{\nabla}{\|\nabla\|}$.

Preserves gradient direction, caps its magnitude

Summary

- Multi-layer perceptrons automatically learn good features and transformations from data.
- Training loop: forward pass $\rightarrow$ loss $\rightarrow$ backward pass $\rightarrow$ weight update, repeat.
- Backpropagation reuses intermediate computations to efficiently evaluate all gradients via the chain rule.
- Dead ReLU neurons (pre-activation always negative) get zero gradient and stop learning; careful initialization helps.
- Vanishing/exploding gradients arise from multiplying many small (or large) factors across layers.

Reference: weight initialization

- If all weights start at 0, every neuron computes the same thing → no learning
- If weights are too large, activations saturate or explode
- If weights are too small, signals shrink to zero across layers

Goal: keep the variance of activations roughly constant across layers.

- **Xavier initialization** (for sigmoid / tanh): $W_{ij}^{\ell} \sim \mathcal{N}\left(0,\ \frac{1}{n_{\ell-1}}\right)$

  where $n_{\ell-1}$ = number of inputs (fan-in) to the layer

- **He initialization** (for ReLU): $W_{ij}^{\ell} \sim \mathcal{N}\left(0,\ \frac{2}{n_{\ell-1}}\right)$

  factor of 2 compensates for ReLU zeroing out half the inputs